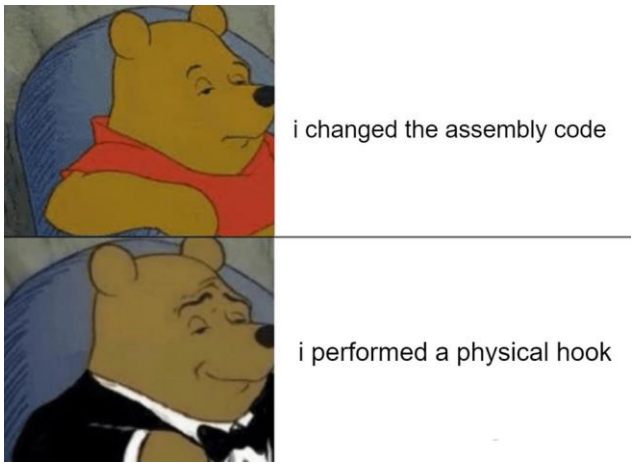
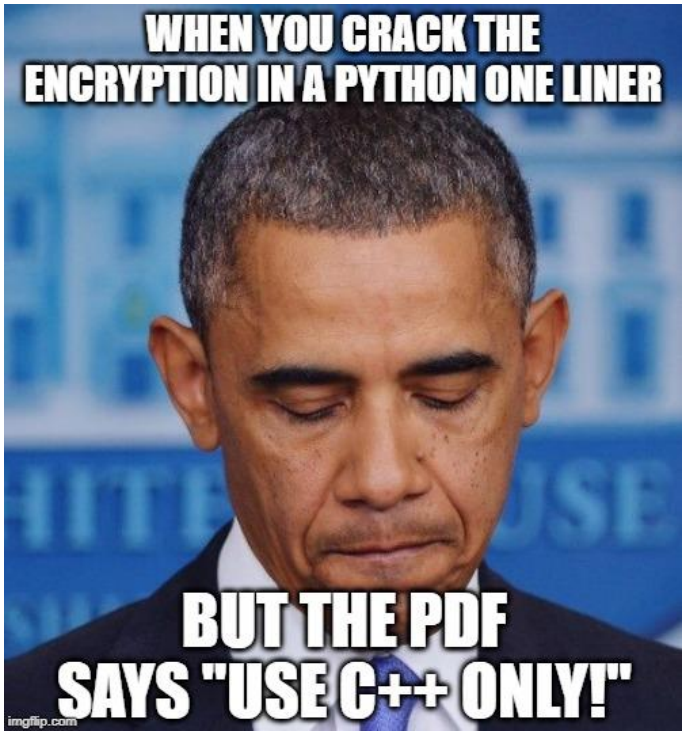
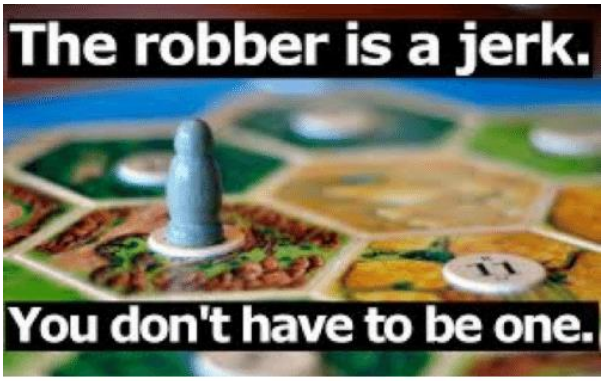
תרגיל בית 3



when you upload a hook and get 'Bad Zip File'





Settlers Of Catan Memes

חלק יבש

1. תחילה נמצא את הכתובת בזיכרון הגלובאלי אליה כותבת התוכנית את מספר המילישניות ע"י מדידת הזמן בין שתי גישות לשרת בריצה רגילה ומעבר על כל הכתובות אליהן ניגשת הפונקציה poll עד שנמצא ערך מתאים למדידה. לאחר מכן נריץ תהליך נפרד שירוץ בלולאה אינסופית ויכתוב לאותה כתובת את הערך במילישניות שברצוננו לגרום לתוכנית לחכות. זיכרון גלובאלי ניתן לגישה ע"י כל תהליך, לכן זה ניתן לביצוע.

2. נשכפל את הפונקציה המקורית כך שיהיו לנו שני עותקים שלה בזכרון - הפונקציה המקורית והפונקציה המשוכפלת. הפונקציה המשוכפלת תהיה זהה לחלוטין לפונקציה המקורית ובסה"כ תקבל את הקלט מכל אחת מנקודות הכניסה האפשריות ותחזיר את הפלט המצופה. בפונקציה המקורית נבצע הוק בסוף שיבצע מספר פעולות. ראשית, נבין מאיזו נקודת כניסה נכנסנו לפונקציה - ניתן לבצע זאת ע"י גישה לכתובת החזרה שנמצאת סמוך ל-ebp וחיסור גודל פקודת call, וכך למעשה נגיע לפקודה שקדמה לכתובת החזרה שהיא בעצם הקריאה לפונקציה. ע"י פענוח הקידוד של הקריאה לפונקציה, מאחר ואנחנו יודעים את ה-opcode של call, ניתן להבין מאיזו נקודת כניסה נכנסנו לפונקציה. כעת כשאנחנו יודעים את נקודת הכניסה אנו יודעים בדיוק איזה חישוב יבוצע על הקלט ונוכל לטפל בפלט. בשלב זה נבצע קריאה לפונקציה המשוכפלת שיצרנו מנקודת כניסה התואמת לנקודה שממנה נכנסנו לפונקציה המקורית, ואילו הקלט יהיה בדיוק היפוך הביטים של הקלט המקורי. כלומר, אם נכנסנו לפונקציה המקורית עם קלט x מנקודת כניסה כלשהי, נקרא לפונקציה המשוכפלת עם הקלט שהוא היפוך הביטים של x ומאותה נקודת כניסה. הפונקציה המקורית תקבל את ערך החזרה של הפונקציה המשוכפלת ותחזיר את סכום הערכים, כך שתמיד ערך החזרה של x יהיה זהה לערך החזרה של היפוך הביטים שלו. בנוסף, על מנת לא לדרוס פקודות שיתכן ונמצאות בזכרון לאחר ret, נבצע את כל הלוגיקה של ההוק שלנו באיזור אחר בזכרון, כאשר נדרוס את פקודת ret המקורית בפקודת jmp אל ההוק ומשם נבצע ret.

3. נדרוס את תחילת הפונקציה ונשתול במקום קפיצה לפונקציה שלנו שתפענח את הפורמט ושאר הארגומנטים (אפשר ידנית, ואפשר לקרוא ל-sprintf שתעשה זאת בשבילנו), נצפין את המחרוזת שקיבלנו ונרצה לשלוח אותה חזרה אל sendf המקורית, בשלב הזה נשחזר את הפקודות שדרסנו מ-sendf בגוף הפונקציה שלנו, נכניס במקום ארגומנט הפורמט "%s" וארגומנט יחיד נוסף אחריו שיצביע למחרוזת המוצפנת שיצרנו. כעת נקפוץ חזרה ל-sendf.

4. בתחילת connect נשתול קפיצה לפונקציה שלנו, שתקרא ל-CreateProcess ותתחיל לבצע את parse בתהליך מקביל. לאחר מכן נשחזר את הפקודות שדרסנו מ-connect ונקפוץ חזרה אליה. בנוסף נצטרך לטפל בקריאות הרגילות ל-parse שלא תקראנה. זאת נפתור ע"י קפיצה בתחילת parse אל פונקציה נוספת שלנו שתמיד תצא. הפונקציה שנקרא לה בCreateProcess לא תהיה parse הרגילה אלא פונקציה נוספת(!) שלנו שתבצע את הפקודות שדרסנו מ-parse הרגילה ואז תקפוץ אל הנק' בה ניתן להמשיך את הפונקציה המקורית, כך שלמעשה נקודת הכניסה החדשה של parse תהיה רק דרך connect ואילו נקודת הכניסה הרגילה שלה תוביל לסיום מיידי.

5. באופן כללי נרצה לעטוף כל קריאה ל-calc בפונקציה שלנו שתדפיס את ערך ההחזרה. מכיוון שלא הגיוני לשכתב כל קריאה ל-calc נצטרך לקפוץ בתחילתה מה שיגרום לבדיקה העצמית של הקוד של calc להכשל ולהחזיר ערכים לא מוגדרים. לכן ניצור עותק של הפונקציה במקום אחר בזיכרון ש-לו נקרא דרך פונקציית המעטפת שלנו שלאחר ריצת העותק תדפיס את ערך ההחזרה ותחזיר אותו ובכך תשמר את ההתנהגות הנורמלית של calc. הבדיקה העצמית של עותק calc תצליח כיוון שהקוד שלו נותר זהה למקור. פונקציית calc המקורית תהווה רק כתובת לקריאה לפונקציית המעטפת שלנו.

.6

א. נבצע הוק בתחילת הפונקציה שייצור מעין קריאת מעטפת לפונקציה המקורית. כלומר, תחילתה של הפונקציה תהיה קפיצה לקוד שתול שלנו, שבו נריץ את הלוגיקה של הפונקציה המקורית ונקבל את ערך החזרה שלה. אנו רוצים שערך החזרה בסוף הרקורסיה יהיה גדול פי 2, ולכן בכל שלב ברקורסיה כאשר הקוד השתול מקבל את ערך החזרה נכפיל אותו ונקפוץ חזרה לפונקציה המקורית במיקום המתאים, כך שתימשך הקריאה הרקורסיבית עם ערך החזרה המוכפל.

ב. נבצע באופן זהה לחלוטין, רק שהפעם נצטרך להבחין מתי יש לכפול את ערך החזרה. כלומר, נצטרך להחזיק משתנה גלובלי שיהיה נגיש מכל שכבות הרקורסיה ויאפשר לכל שכבה לדעת האם היא השכבה החיצונית ביותר או לא. ניתן לבצע זאת ע"י החזקת קאונטר שיספור את עומק הרקורסיה. כלומר, בתחילת כל קריאה רקורסיבית הקאונטר יגדל ב-1 ובסופה הוא יקטן ב-1. כך, ניתן לזהות את השכבה החיצונית ביותר לפי השוואת ערך הקאונטר ל-1 (מאחר וכל קריאה רקורסיבית שהגדילה אותו כבר סיימה לרוץ והקטינה אותו בחזרה, ואילו הקריאה החיצונית הגדילה אותו אך עוד לא הקטינה) ובעת זיהוי שכבה זו נכפיל את ערך החזרה ב-2 ונחזירו.

חלק רטוב

- **חלק ראשון:**

התוכנית keygen.exe יצרה מפתחות על פי החוקיות הבאה: עבור כל תו ממחרוזת המקור החסר 0x20 מערך ה-ascii שלו (נסמן מספר זה ב-i) והחזר את התו ה-i במערך הבא:

{., 5, `, >, A, d, I, G, -, *, 8, E, e, }, @, B, D, R, F, k, z, w,], Z,
Y, g, r, s, m, 1, q, \, 2, |, 3, J, j, (, K, u, X, , !, ', :, _, v, 6, a,
W, f, {, y, ", C, ;, +, L, <, S, 4, H, O, [, 0, ?, $, b, h, %, /, M, t,
o, T, 9, ,, N,), &, l, =, x, U, p, V, ^, 7, ~, n, #, c, i, Q, P, ð, û, ` }

מערך זה נמצא נגיש בזכרון התכנית ועל כן הצלחנו בקלות לקודד אותו ב-CPP.

לכן, כתבנו תוכנית CPP פשוטה שמבצעת את התהליך ההפוך, כלומר, לכל תו מהקלט: מצא את האינדקס שלו במערך והוסף 0x20.

בשיטה זו מצאנו שהסיסמה הגישה לאתר LHSBXVUGS1KFFADR היא מפתח של המחרוזת Y][/Hus'[=F22$01 שהיא סיסמת הגישה לדף Tools באתר.

- **חלק שני:**

בחלק זה התבקשנו לפענח את ההודעה האחרונה שהושארה השרת ע"י הרצת פקודת DMESG בתוך תוכנית client.exe. תחילה תכננו לדלג על ההצפנה ע"י הוק ב-client שישלח את ההודעה באמצעות pipe לא מוצפן אבל אין לנו שליטה על הצורה בה השרת מחזיר תשובה אז ויתרנו על האפשרות הזאת. החלטנו לחקור את securepipe.exe ואולי לשתול הוק בו אבל ישנן עוד תוכניות שמשתמשות בו וייתכן ונפגע בתפקודן. לבסוף בחרנו לשתול הוק על client.exe ולפענח בזמן ריצה את ההודעה המוחזרת.
לאחר חקר securepipe.exe הערכנו את סדר פעולות ההצפנה. לכל תו המיוצג ע"י שני בתים, ממירים כל בית לפי ערכו בחפיסת קלפים (ללא 10) כאשר כל ערך שאין לו "ערך קלף" מפצלים לפעולה חשבונית של חיבור או חיסור. כלומר,

0	1	2	3	4	5	6	7	8	9	(A)10	(B)11	(C)12	(D)13	(E)14	(F)15
x+y=0	A	2	3	4	5	6	7	8	9	J	Q	K	x+y=13	x+y=14	x+y=15

אז לתו '0' שערכו 0x30 נקבל '4-34' ועבור התו 'q' שערכו 71 נקבל '7A'.
כדי לבצע את הפענוח בזמן המתאים, מצאנו שהפונקציה recv נמצאת בdll ws2_32 כך שנוכל שבצע IAT Hook ולבצע קוד שלנו בלי צורך לשחזר פקודות. מצאנו את הכתובת שלה בטבלת הimports בIDA ודרסנו את הרשומה המתאימה כך שתקפוץ לפונקציה שלנו בעזרת DLL Injection. הפונקציה שלנו קוראת ל-recv מפענחת את הפלט שלה ומחזירה את המחרוזת המפוענחת במקום הפלט של recv.
התכנון היה שכאשר השרת יריץ את ה-toolfix שנעלה יהיה זה ה-DLLInjector שיפעיל את client.exe עם ארגומנט DMESG, כלומר יבצע את הפקודה ואז כשיקבל תגובה מוצפנת מהשרת יפענח וידפיס את ההודעה כשהיא מפוענחת.

- **חלק שלישי:**

בהודעה שקיבלנו בסוף החלק הקודם צויין כי השודד נמצא במשבצת B0 ועל מנת לשחרר אותו יש למצוא קוד שקשור ל-ROBBER_CAPTURED ובהזנתו נקבל שבע. התחלנו ממעבר על הקבצים שנמצאים ב-Tools וניסינו להבין אילו מהקבצים יהיו רלוונטיים למטרה שלנו. ממעבר על Codes, מצוין כי הכלי משמש להחלפת קוד קיים במערכת בקוד חדש. לכן, הסקנו שעם Hook מתאים נוכל להשיג כך גישה לקוד קיים.

התחלנו מניתוח סטטי של הקובץ וראינו כי הוא מקבל שני ארגומנטים – הראשון הוא מזהה של קוד והשני הוא קוד ישן הקשור לאותו מזהה, והתכנית תפלוט בהתאם קוד חדש עבור המזהה הנתון במידה והקוד הישן שסופק הוא נכון.

לכן, הכיוון הראשוני שלנו היה לנסות להבין איזה חלק בקוד מאמת את הקוד הישן ולנסות לעקוף את האימות. לאחר חקירת הקוד, ראינו שלמעשה מדובר בקוד שהוא מעטפת לקוד פייתון. התהליך הראשי של codes.exe יוצר תהליך בן ע"י CreateProcessA, פותח pipe ע"י CreatePipe ומעביר אליו פקודות פייתון כאשר התקשורת בין ה-threads היא על גבי ה-pipe שנפתח.

בעקבות כך, ניסינו קודם כל להבין מהן הפקודות שיבוצעו וסרקנו את הקוד במטרה לזהות טעינה של פקודות. זיהינו שמתבצע import למודול של database, ככל הנראה Django, ומבצעים בעזרתו שאילתות על הדאטה-בייס. השאילתות שזיהינו הן:

```
codes = Code.query.filter_by(code = '%s.('all();

Print('NO SUCH CODE' if len(codes) <= 0 else codes[0.[event.key)


codes = [code for code in Event.query.filter_by(key = '%s.('first.()codes if not code.used];

print('' if len(codes) <= 0 else codes[0.[code)
```

מקריאה של השאילתה הראשונה, ניתן להבין כי שאילתה זו בוחנת את מזהה הקוד שהתקבל כארגומנט הראשון, מסננת מתוך אוסף הקודים את הקוד שהתקבל כארגומנט השני ומחזירה את המזהה המשויך ל-event שלו. אם לא נמצא הקוד שהוזן, תודפס ההודעה .NO SUCH CODE

השאילתה השנייה מסננת לפי המזהה שמתקבל בארגומנט השני ונותנת את הקודים הרלוונטיים עבורו.

מההודעה שקיבלנו עם סיום החלק הקודם בתרגיל, הבנו כבר שה-event שעלינו להגיע אליו הוא ROBBER_CAPTURED כך שהוא יהיה הארגומנט הראשון, ונותר לנו כעת רק למצוא קוד המשויך אליו.

לכן, החלטנו להתמקד בשאילתה הראשונה ולנסות לעקוף את וידוא הקוד הישן, וכך כאשר נגיע לשאילתה השנייה, היא תורץ עם המזהה שנזין כקלט (ROBBER_CAPTURED) ויודפס לנו קוד חדש.

השתמשנו בהוק דינאמי על מנת למצוא קוד שמספק את השאילתה. יצרנו dll ו-injector שעליהם נפרט.

שאילתה זו נטענת בכתובת 0x401410 כאשר כתובת הבסיס היא 0x400000, כלומר, ב-offset של 0x1410 מתחילת התכנית, ולכן ה-injector מגדיר את hooked_function לפי אופסט זה. בעת ריצתו, ה-injector, בדומה לדוגמאות שראינו בכיתה, קורא ל-CreateProcessA עבור Codes.exe כאשר הארגומנט הראשון, כאמור, הוא ROBBER_CAPTURED והארגומנט השני הוא מחרוזת שערכה לא משנה היות ומטרת ההוק לעקוף אותה ומאחר שאיננו יודעים קוד בשלב זה. לאחר יצירת התהליך, ה-injector טוען את ה-dll שיצרנו ב-thread נפרד ע"י .CreateRemoteThread

בנוסף, יצרנו dll שיסייע לנו במעקף השאילתה ובמציאת סיסמה. כמו שפירטנו קודם לכן, בהינתן מזהה קוד כלשהו, השאילתה תוודא את נכונות הקוד שסופק ותאימותו לשאילתה ובמידה שהצליחה תדפיס את המזהה. לכן, על מנת לעקוף את שלב האימות החלטנו לשנות את השאילתה באמצעות ה-Hook. על מנת לא לפגוע בהמשך הפעולה התקין של התכנית, ולדמות מצב שבו הצליחה השאילתה, עלינו לדאוג להדפסה מתאימה. לכן, מספיק לנו להחליף את השאילתה הראשונה בשאילתה שתבצע הדפסה של המזהה הרצוי של ה-Event, שהוא כאמור ROBBER_CAPTURED. לכן, יצרנו ב-dll מחרוזת המכילה את השאילתה החדשה שהיא למעשה 'print('ROBBER_CAPTURED';. בנוסף, בסוף השאילתא שמנו את התו '#' על מנת להתעלם מכל פקודה שיכולה להמצא אחריה ע"י כך שתחשיבה כהערה, ובכך נמנע

משגיאת פייתון. כעת, כתבנו פונקציה פשוטה שנקראת CreateQuery שרק מחזירה את המצביע למחרוזת שמכילה את השאילתה, כך שהמצביע ימצא ברגיסטר eax. בתחילת פעולת ה-dll, תיקרא פונקציה זו על מנת לשמור את המצביע לשאילתה ב-eax, לאחר מכן תשוחזר פעולת הפונקציה המקורית שאותה דרסנו למעט טעינת השאילתה שבה נטען את השאילתה שלנו מתוך eax. לאחר מכן, נקפוץ לפקודה המופיעה בפונקציה המקורית לאחר טעינת השאילתה, שנמצאת בכתובת 0x401421, כלומר 0x11 מתחילת הפונקציה. לכן נגדיר את האופסט לקפיצה בהתאם כך שנמשיך את פעולת הפונקציה המקורית כרגיל.

לאחר המשך ניתוח הקוד, ניתן לראות כי ישנו לאחר יצירת תהליך הבן, ישנה קריאה לפונקציה SetHandleInformation שבה מגדירים את ה-handlers של תהליך הבן עבור i/o בתור ה-handlers i/o של ה-pipe, כך שהקלט והפלט של תהליך הבן יעברו דרך ה-.pipe

נשים לב שכעת במהלך ביצוע השאילתה השנייה, הפלט יהיה קוד חדש שאותו אנו מחפשים, אך הפלט כאמור יועבר ל-pipe ונצטרך לגשת אליו. נשים לב כי הקריאה מה-pipe מתבצעת ע"י ReadFile אל תוך באפר וברצוננו לקרוא את תוכן הבאפר. הפונקציה ReadFile נטענת חיצונית מ-KERNEL32.dll, ולכן נוסיף ל-Hook הקיים שלנו פונקציה נוספת שמבצעת IAT Hook על ReadFile. יצרנו פונקציית SetHook שמוצאת ב-dll את ReadFile ע"י שימוש ב-GetProcAddress ובנוסף יצרנו פונקציית FakeReadFile שבעת הפעלת ה-Hook קוראת ל-ReadFile האמיתית ובנוסף מדפיסה את הבאפר לאחר הכתיבה אליו. כעת, החלפנו את הכניסה של ReadFile ב-IAT לכתובת של FakeReadFile וכך בעת הקריאה ל-ReadFile לשם קריאה מתוך ה-pipe, תיקרא הפונקציה המזויפת שלנו שבנוסף לקריאה מה-pipe וכתיבת הפלט לתוך הבאפר – תדפיס את ערכו, שהוא למעשה הקוד המבוקש.

לאחר יצירת ה-dll וה-injector המתוארים להלן, העלנו אותם בפורמט הנדרש מכווצים ל-zip במקום Codes המקורי ואכן הודפס לממשק הקוד שרצינו המשוייך ל-ROBBER_CAPTURED והוא .UVWN9TF1XB

כעת, עלינו לדאוג לכך שבעת הפעלת ה-Hook הדינאמי שיצרנו הטלת הקוביות תיתן 7 וכך ישוחרר השודד לחופשי. לשם כך, עברנו בשנית על הכלים המופיעים ב-Tools וראינו כי Dice שולט על הטלות הקוביה והחלטנו לבצע עליו Hook. הפעם ניתן לראות כי פורמט ההעלאה אינו zip אלא קובץ executable יחיד, כך שההוק שנבצע הפעם יהיה פיזי.

מניתוח הקובץ dice.exe, ניתן לראות שהוא מסוגל לקבל כארגומנט קוד, ובמהלך התכנית ניתן לראות שמתבצעת השאילתה הבאה:

```
print(Code.query.filter_by(code = '%s.('first.()event.key
```

בהמשך נבין את ההקשר שלה.

במהלך הקוד ניתן לראות כי מוגרל מספר בין 2 ל-12 שמייצג את סכום הטלת שתי הקוביות, ולאחר מכן בהתאם לסכום שהתקבל מוגרל הערך בכל קובייה כך שסכום הערכים יתאים לסכום שהתקבל.

ניתן לראות כי לאחר הגרלת סכום הקוביות, נשמר הערך שמתקבל על המחסנית. לאחר מכן, בדומה ל-Codes.exe, נפתח שוב תהליך בן ו-pipe לקלט ופלט ושם רצה השאילתה שראינו קודם לכן, ומסננת למעשה את ה-Event התואם לקוד שהוזן בקלט.

בשלב זה אנו כבר יודעים את תוצאת סכום הקוביות, ולכן כאן מיקמנו את ה-Hook שלנו, על מנת לדאוג שבמידה והוזן הקוד המתאים, שאותו מצאנו בשלב הקודם ע"י Hook דינאמי, ערך הקוביות יהיה 7. כך בעצם נוודא שברגע שהוזן הקוד הנכון, ערך הקוביות בהטלה הבאה יהיה 7 – מה שיגרום לשחרורו המיידי של השודד.

לשם ביצוע ה-Hook, כמו שראינו בכיתה, בחרנו ליצור קפיצה מסופה של פונקציה אל קטע קוד ששתלנו. ראשית, השתמשנו בתוכנה לעריכת קבצי PE על מנת להוסיף סגמנט חדש עם הרשאות הרצה כך שנוכל לכתוב בו קוד להרצה. לאחר מכן, מיקמנו את ה-Hook בסוף הפונקציה המקורית ע"י כך שדרסנו את אחת הפקודות בסופה עם jmp אל קטע הקוד שכתבנו בסגמנט החדש. נשים לב כי לא בהכרח ניתן לדרוס דווקא את הפקודה האחרונה (במקרה הזה ret), כי הקידוד של פקודת הקפיצה אל ה-Hook יכול להיות גדול מהפקודה שדורסים. עם ההגעה לקפיצה, בדקנו ראשית את המקרה שבו אין ארגומנט לתכנית - כלומר, לא סופק קוד. במקרה זה נקפוץ חזרה לסוף הפונקציה המקורית מאחר וכל הלוגיקה ההכרחית כבר בוצעה ועלינו להיות שקופים.

לאחר מכן, נטפל במקרים בהם סופק ארגומנט (קוד). במצב זה נדחוף על המחסנית מחרוזת המייצגת את הקוד שקיבלנו בשלב הקודם (UVWN9TF1XB) ונשווה אותו לקוד שסופק בארגומנט. במידה והם שונים, נשחזר את הרגיסטרים והמחסנית ונקפוץ חזרה לסוף הפונקציה המקורית כך שיתקבלו הקוביות שנבחרו לפני הקפיצה ל-Hook. במידה והם שווים, נעדכן על המחסנית את סכום הקוביות ל-7 ונעדכן ברגיסטר eax את תוצאות שתי הקוביות ל-4 ו-3 כך שיתאימו לסכום 7. נשחזר את הרגיסטרים והמחסנית ונקפוץ חזרה לסוף הפונקציה המקורית וערכי הקוביות יהיו 3 ו-4.

בכל המקרים דאגנו לשחזר את הרגיסטרים הרלוונטיים והמחסנית, ובנוסף דאגנו שרק במקרה שבו הקוד שהוזן תקין ישתנו ערכי הקוביות וביתר המקרים ערכיהן ישוחזרו חזרה לתוצאה האקראית שהוגרלה כך שה-Hook שקוף במידת האפשר ובו בזמן ידאג לקבלת 7 עם הזנת הקוד כך שהשודד ישוחרר לחופשי.